Ejercicio de usabilidad de Ghosts

Ricardo Jacas

13 de diciembre de 2013

Problema 1, Cuentas Internacionales

Se desea hacer un programa que represente la interacción de un usuario con su cuenta bancaria, en cajeros automáticos que manejan distintos tipos de moneda.

Para ello debe crear las clases `User` y `Account` que representan al usuario y a su cuenta, respectivamente. Además se cree la clase `ATM` que representa a los cajeros automáticos. Por simplicidad, el sistema sólo debe manejar Dólares y Pesos, mediante las clases `Dollar` y `Peso`.

Cada `User` podrá retirar dinero, en algún tipo de moneda, mediante su método `getMoney`, que extraerá dicho dinero de un `ATM`, indicando también de que cuenta (`Account`) y cuanto quiere retirar, expresado como `Double`.

Las cuentas se crearán para un tipo de moneda específica y un usuario en particular. La cantidad de dinero que almacenan debe ser representada por un tipo de moneda, no por un número.

Los `ATM` también deben estar asociados a un tipo de moneda en específico, pero deben permitir la extracción de dinero de cualquier cuenta. Lo anterior mediante el método `getMoney`, que dado un usuario y una cuenta específicos, junto con una cantidad de dinero representada por un número (en `Double`), retornará el dinero solicitado, en la moneda que maneja.

No debe crear jerarquías de `ATM` o de `Account`, sólo preocúpese que ambas clases manejen los dos tipos de moneda.

Escriba todos sus números en formato `Double` (ejemplo: `0` es `0.0`).

Para sus pruebas, cree la clase `TestAccounts` que extienda de `TestCase` y escriba los siguientes métodos de prueba:

- `testCreation`: Para probar los constructores de sus clases y los valores iniciales.
- `testGetMoney`: Para probar el retiro de dinero por parte de un usuario, de su cuenta, en un cajero que utiliza la misma moneda que su cuenta.
- `testGetTonsOfMoney`: Para probar el retiro de dinero por parte de un usuario, de su cuenta, en un cajero que utiliza la misma moneda que su cuenta, pero pidiendo más dinero del que su cuenta contiene.
- `testGetForeignMoney`: Para probar el retiro de dinero por parte de un usuario, de su cuenta, en un cajero que utiliza una moneda diferente a la de su cuenta.

Pregunta 2, Sistema de archivos

Un sistema de archivos cotidiano está compuesto por archivos y capetas, que contienen archivos y/o carpetas.

Se desea implementar un sistema de archivos donde cada carpeta tenga ordenados, por nombre, sus elementos. Para ello se implementarán las clases `NFolder` y `NFile`. Por simplicidad ambas clases se representarán por un nombre y tendrán un método `getPath` para indicar su ruta completa desde la carpeta que esté más arriba en el sistema. Además, las carpetas tendrán los siguientes métodos:

- `getSize`: para obtener la cantidad de archivos (`NFile`) que posee.
- `getElem`: que dada una posición, retorna el elemento en dicha posición.
- `addElem`: para agregar un elemento.
- `removeElem`: para quitar un elemento.

Para reforzar que nuestras carpetas sólo puedan contener archivos y carpetas, y que estos estén ordenados por nombre, cada carpeta contendrá internamente una lista, representada por la clase `NList` que se encargará de contener los archivos dentro de la carpeta. **Tenga presente que `NList` no es una clase generica, solamente maneja `NFolder` y `NFile`.**

Para implementar `NList` cree una lista doblemente enlazada, con concepto de sucesor y antecesor, utilizando una clase `NNode` que se encargue de manejar está idea y contener un elemento. Usando lo anterior implemente `NList` con los siguientes métodos:

- `size`: para obtener su cantidad de elementos.
- `get`: que dada una posición, retorna el nodo en dicha posición.
- `add`: para agregar un nodo.
- `remove`: para quitar un nodo.

Para sus pruebas, cree las clases `TestFileSystem` y `TestList` que extienda de `TestCase` y escriba los siguientes métodos de prueba:

Para `TestList`

- `testCreation`: Para probar los constructores de `NList` y `NNode`, y sus valores iniciales.
- `testAdd`: Para probar que se generan los cambios esperados al agregar un nodo.
- `testAddRemove`: Para probar que la lista se mantiene coherente al agregar y quitar un elemento.
- `testAddOrdered`: Para probar si efectivamente los elementos están ordenados dentro de la lista.

Para `TestFileSystem`

- `testCreation`: Para probar los constructores de `NFolder` y `NFile`, y sus valores iniciales.
- `testAddFile`: Para probar que se generan los cambios esperados al agregar un archivo.
- `testAddFileOrdered`: Para probar si efectivamente los archivos están ordenados dentro de la lista.
- `testAddFolder`: Para probar si al agregar una carpeta a otra carpeta todo sigue siendo coherente, esto es, que los métodos `size` y `getPath` funcionan correctamente.

Se le recomienda, por simplicidad, que primero realice todos sus métodos de prueba asumiendo que una carpeta sólo puede contener archivos (`NFile`) y cuando todas sus pruebas (salvo `TestAddFolder`) cumplan con el diseño que espera, hacer la extensión a carpetas dentro de carpetas.